Experiment No. 01 Date Page No. 01

01 Setting up and Basic commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with a appropriate commit message.

Solution:-

To initialize a new Git repository in a directory, create new file, add it to the staging area and commit the changes with an appropriate commit message, follow these steps:

1) Open your terminal and navigate to the directory where you want to create the Git repository.

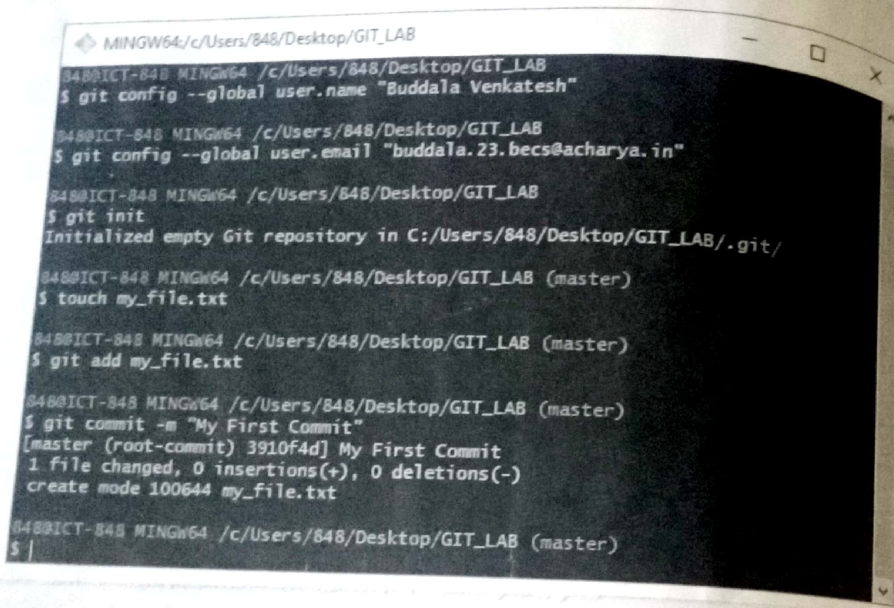
2) Initialize a new Git repository in that directory.

[$ git init]

3) Create a new file in the directory. For example, let's create a file named "my-file.txt". you can use any text editor or command-line tools to create the file.

4) Add the newly created file to the staging area. Replace "my-file.txt" with the actual name of your file:

Teacher's Signature _____ Acharya



Experiment No. _____ Date _____ Page No. 02

[$ git add my-file.txt]

This command stages the file for the upcoming commit.

5) Commit the changes with an appropriate commit message. Replace "your commit message here" with a meaningful description of your changes.

[$ git commit -m "your commit message here"]

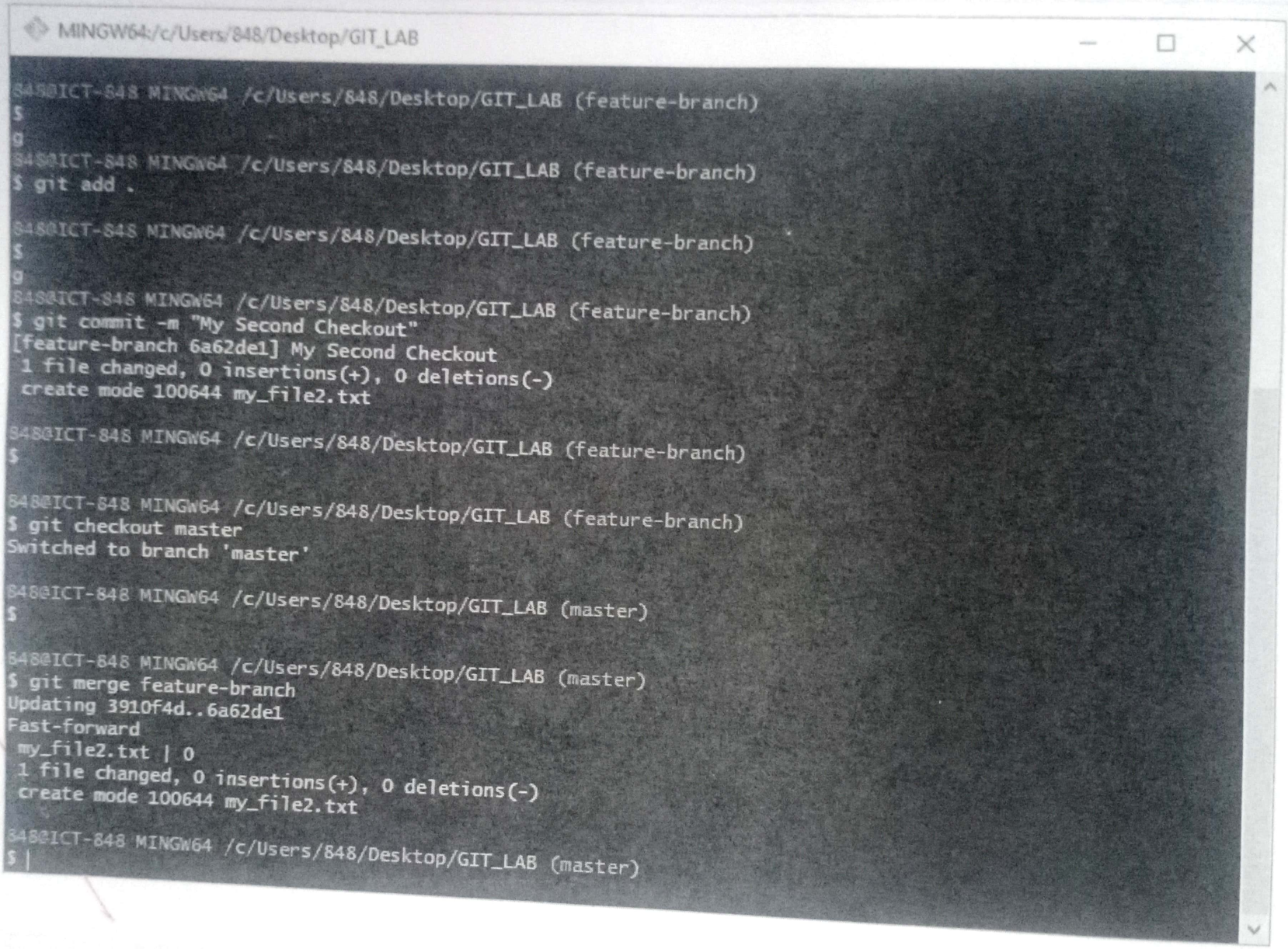
your commit message should briefly describe the purpose & nature of the changes you made

for eg:-

[$ git commit -m "Add new file my-file

After these steps, your changes will be committed to Git repository with the provided commit message. now have a version of the repository with the new and its history stored in Git.

Teacher's Signature _____



02 Creating and managing Branches :-

Create a new branch named "feature-branch". Switch to the "master" branch. merge the "feature-branch" into "master".

Solution :-

To create a new branch named "feature-branch", switch to the "master" branch, and merge the "featured-branch" into "master" in Git, follow these steps:

1) make sure you are in the "master" branch by switching to it:

[$ git checkout master]

2) Create a new branch named "feature-branch" and switch to it

[$ git checkout -b feature-branch]

This command will create a new branch called "feature-branch" and switch to it.

3) make your changes in the "feature-branch" by adding, modifying & deleting file as needed

4) Stage and commit your changes in the "feature_branch":

[$ git add.]

[$ git commit -m "your commit message for feature-branch"]

Replace "your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature_branch".

5) Switch back to the "master" branch.

[$ git checkout master]

6) merge the "feature-branch" into to "master" branch:

[$ git merge feature-branch]

This command will incorporate the changes from the "feature-branch" into "master" branch.

NOW, your changes from the "feature-branch" have been merged into the "master" branch. your Project history will reflect the changes made in both branches.

03 Creating and managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Solution:-

To stash your changes, switch branches, and the apply the stashed changes in Git, you can use the following commands.

1) Stash your changes:

[$ git stash save "your stash message"]

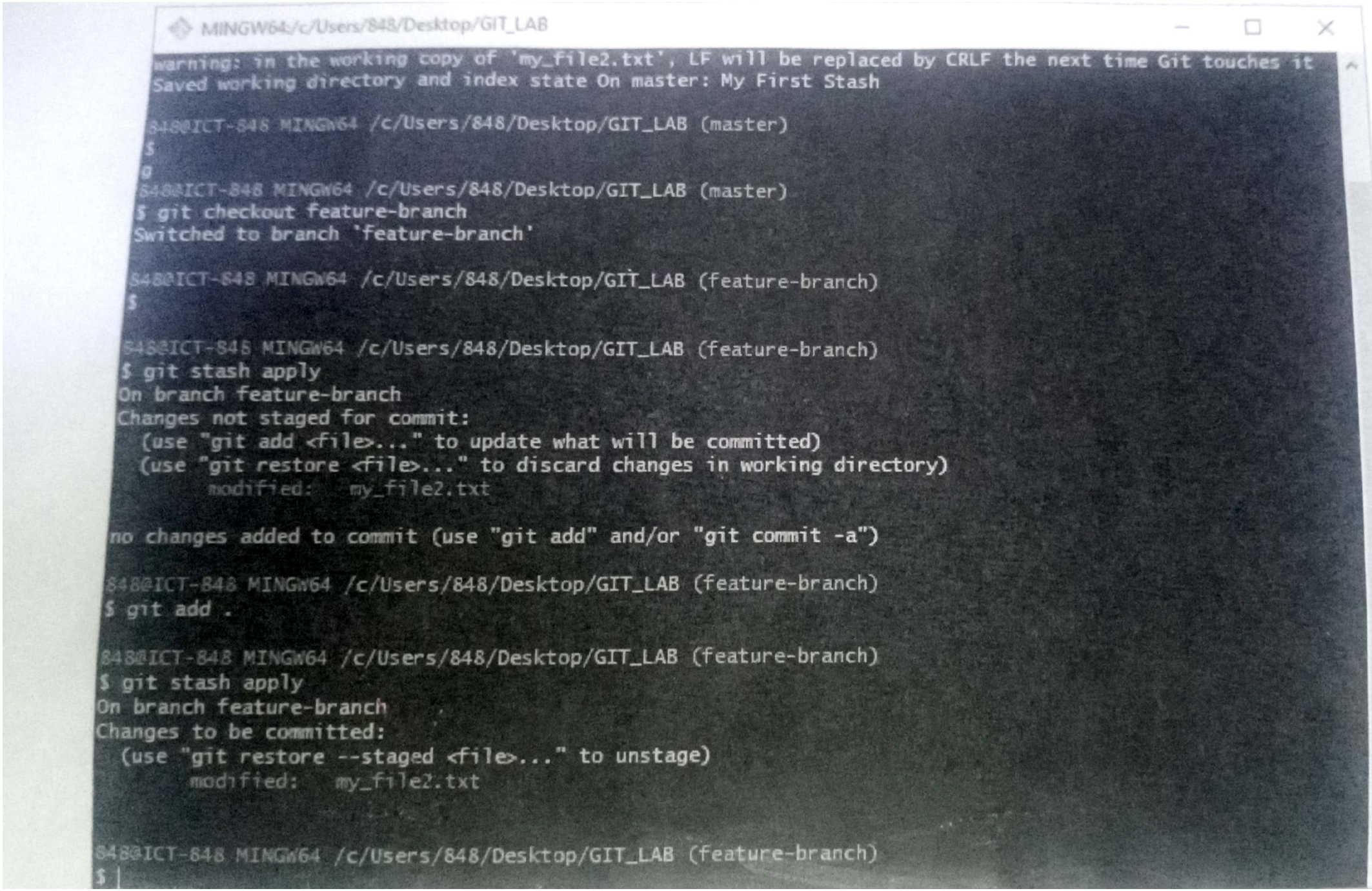
This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2) Switch to the desired branch:

[$ git checkout target-branch]

Replace "target-branch" with the name of the branch you want to switch to

3) Apply the stashed changes:

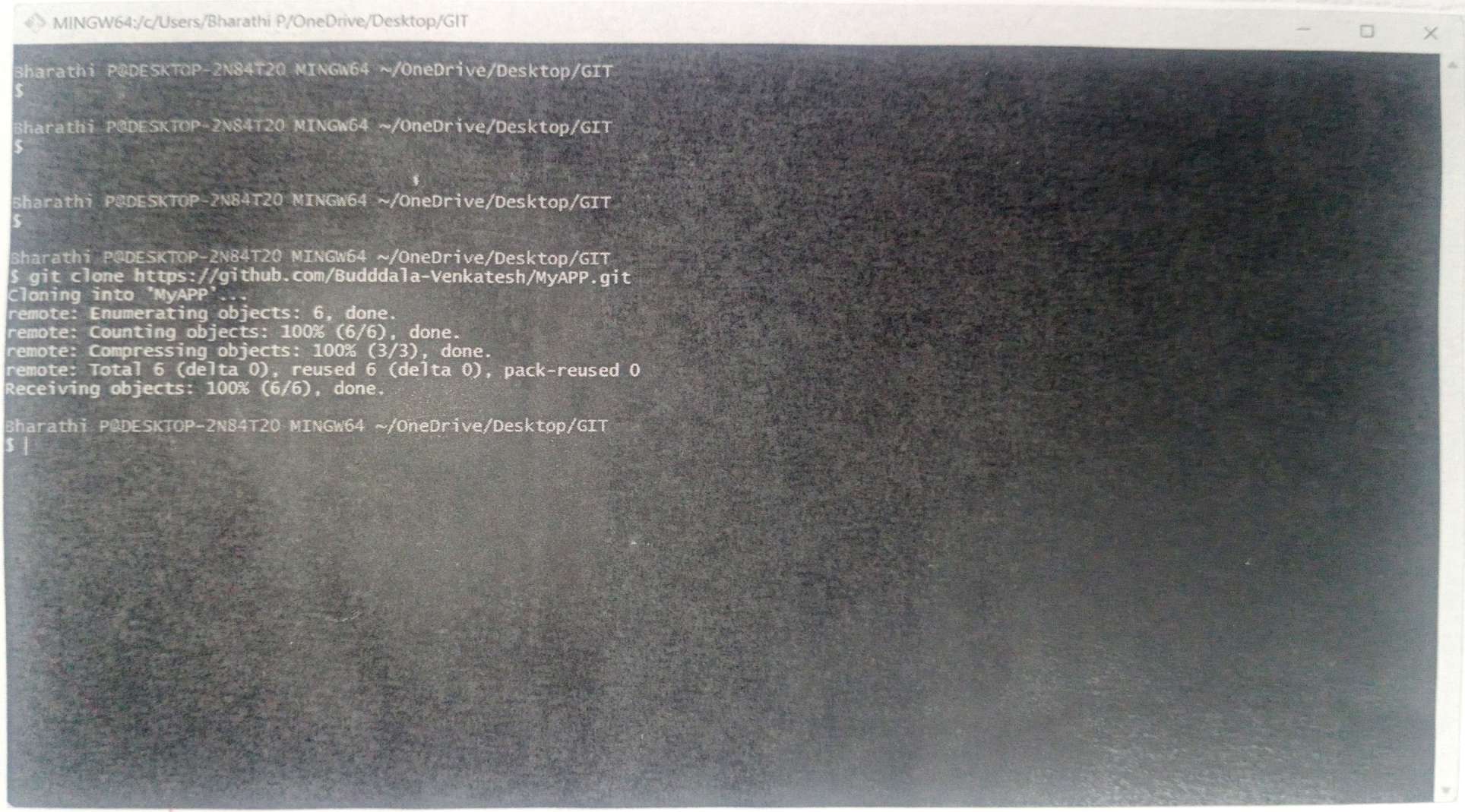


[$ git stash apply]

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (eg: git stash apply stash@{2}) if needed.

if you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.

Remember to replace "your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.



4

04) *Collaboration and Remote Repositories: *

clone a remote Git repository to your local machine.

Solution:

To clone a remote Git repository to your local machine, follow these steps:-

1) Open your terminal or command prompt.
2) Navigate to the directory where you want to the clone the remote Git repository. You can use the cd command to change your working directory.
3) Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone.

For example:- If you were cloning a repository from GitHub, the URL might look like this:

[$ git clone <repository_url>]

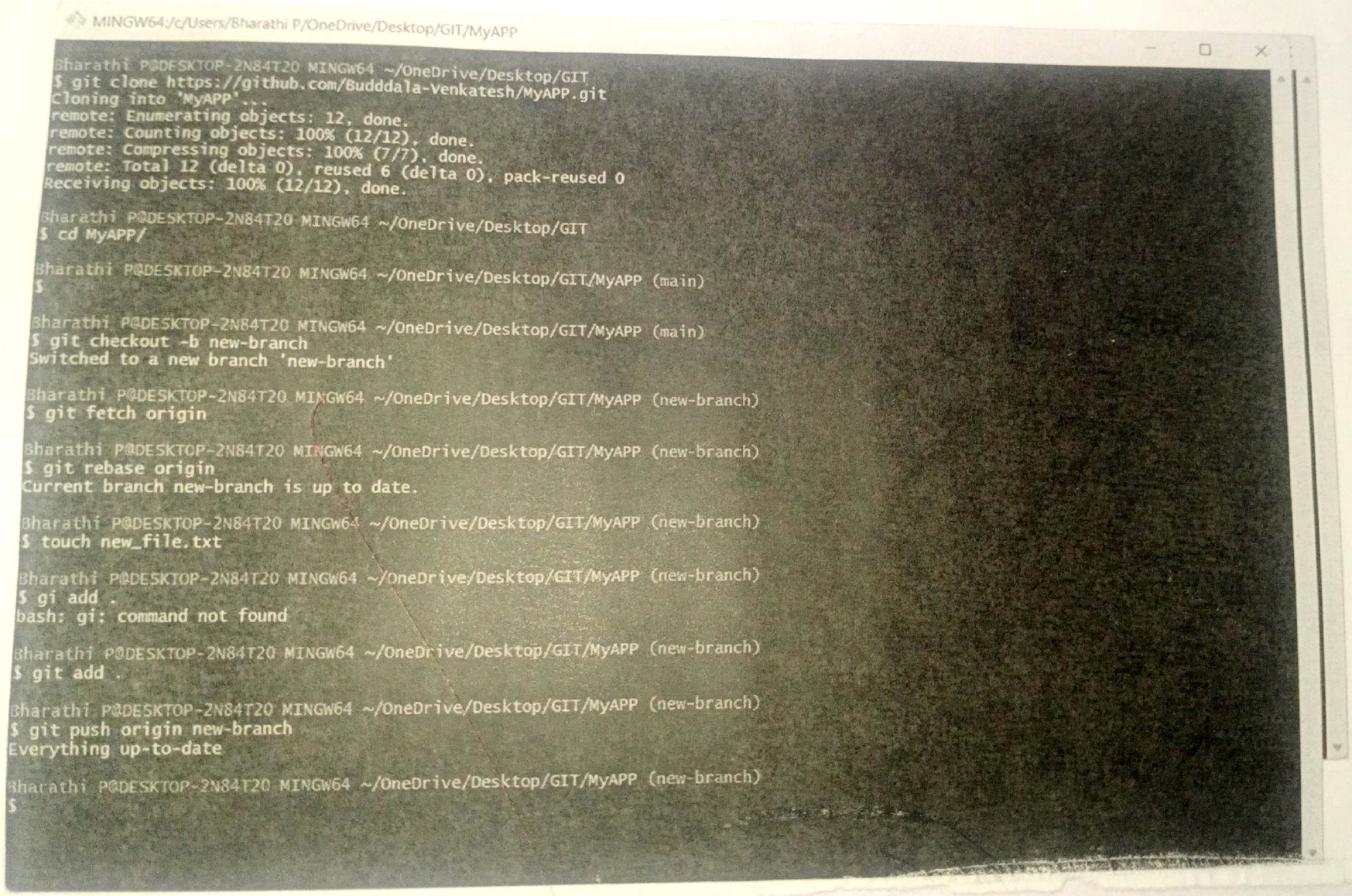
Here's a full example:

[$ git clone https://github.com/username/repo-name.git]

Replace https://github.com/username/repo-name.git with the actual URL of the repository you want to clone.

4. Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.



05 * Collaboration and Remote Repositories *

Fetch the latest changes from a remote repository & rebase your local branch onto the updated remote branch.

Solution :-

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1) Open your terminal @ command Prompt.

2)

3) make sure your are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

[$ git checkout <branch-name>]

3) Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

[$ git fetch origin]

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

4) Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

[$ git rebase origin/<branch-name>]

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

5) Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

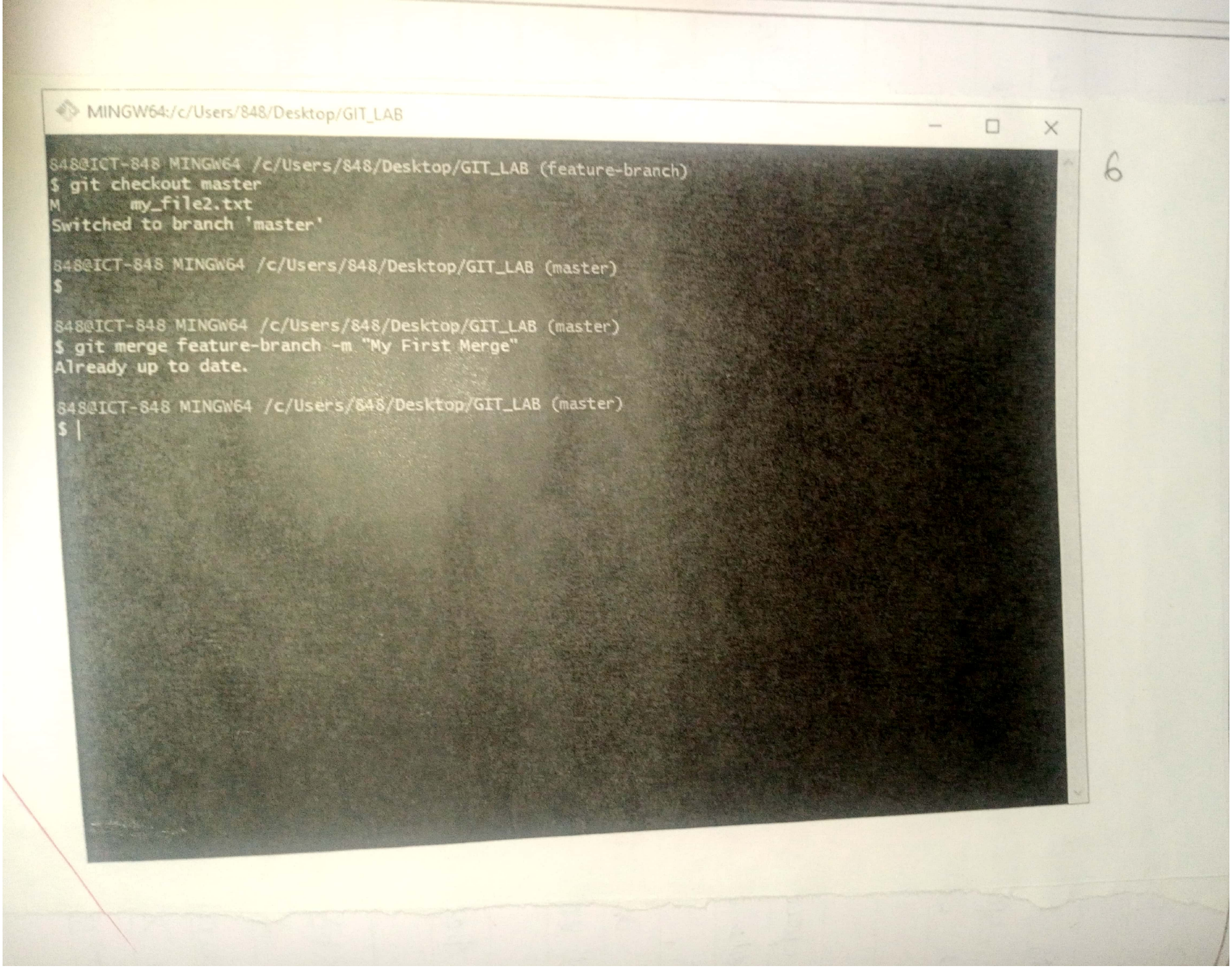
[$ git rebase --continue]

6) After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.

7) If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developer's changes:

[$ git push origin <branch_name>]

Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.



6

Experiment No. 06 Date Page No. 12

06 * Collaboration and Remote Repositories: *

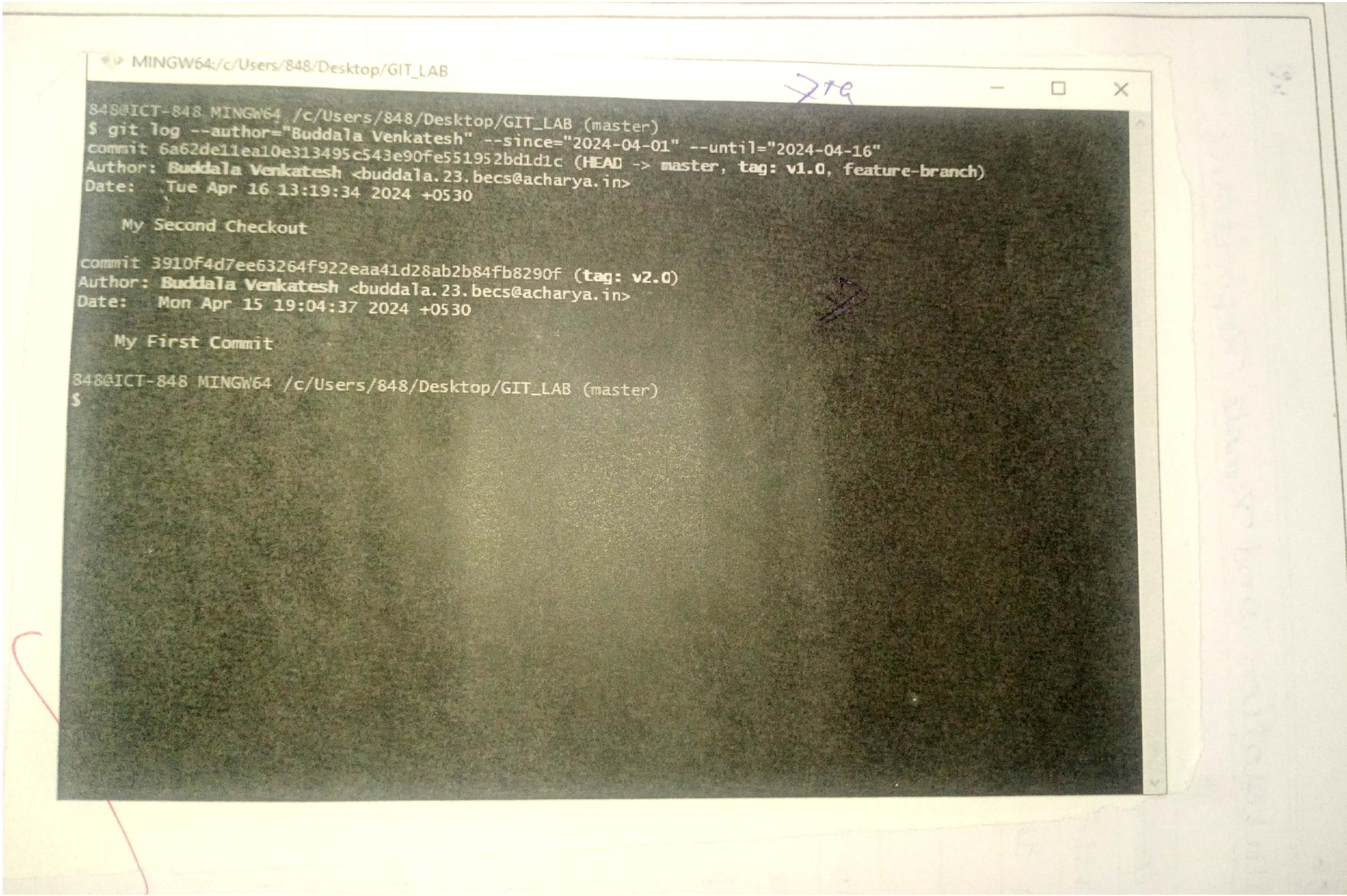
Write the command to merge "feature-branch" into "master" while Providing a custom commit message for the merge.

Solution:-

To merge the "feature-branch" into "master" in Git while Providing a custom commit message for the merge, you can use the following command:

```
$ git checkout master
$ git merge feature-branch -m "your custom commit here"
```

Replace "your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master".



07 * Git Tags and Releases *

Write the command to create a lightweight Git tag named "V1.0" for a commit in your local repository

Solution:

To create a lightweight Git tag name "V1.0" for a commit in your local repository, you can use the following command:

[$ git tag V1.0]

This command will create a lightweight tag called "V1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

[$ git tag V1.0 <commit-SHA>]

Replace <commit-SHA> with the actual SHA-1 hash of the commit you want to tag.

08 * Advanced Git Operations : *

Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

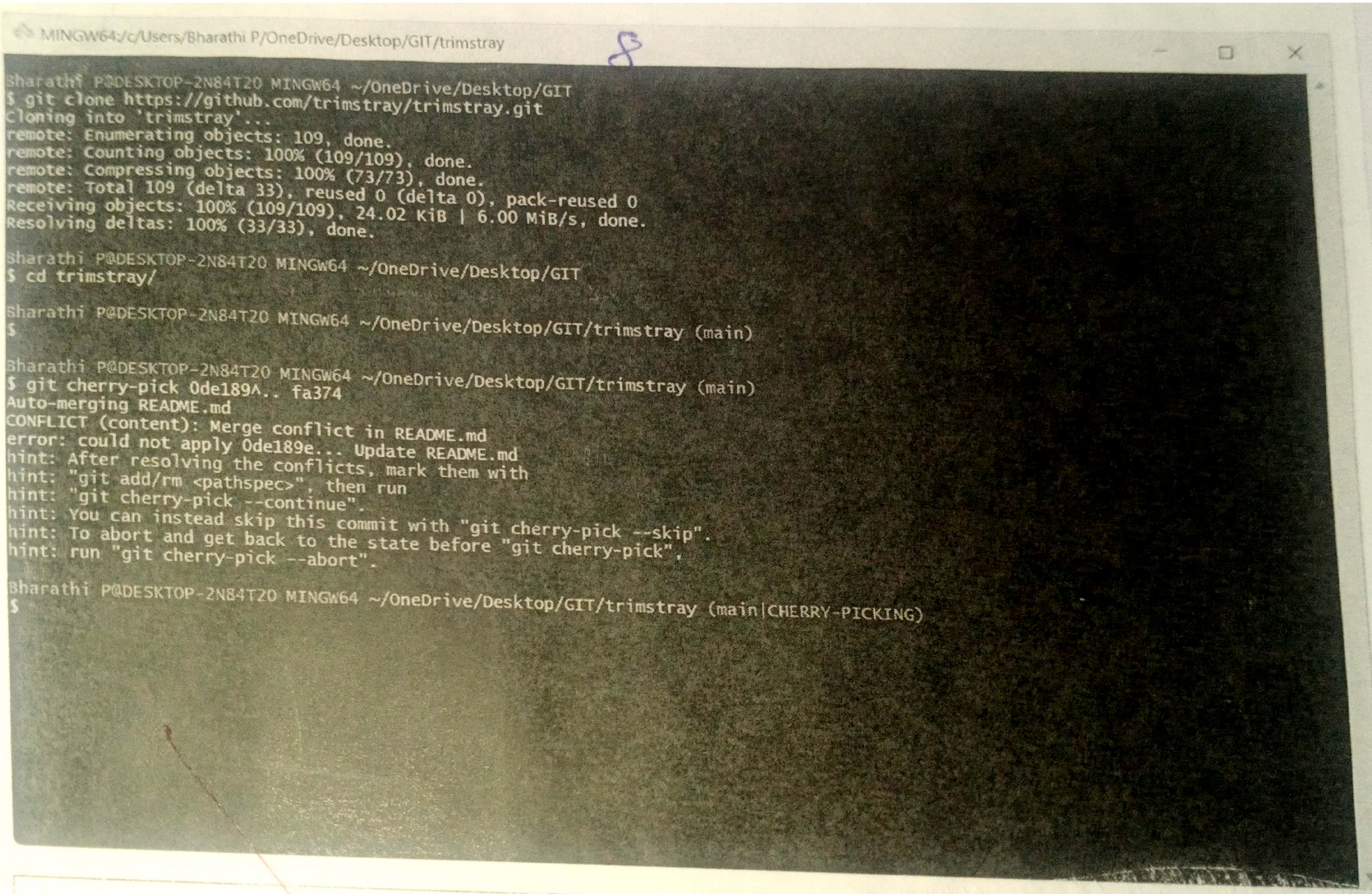
Solution :

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

[$ git cherry-pick <start-commit>^..<end-commit>]

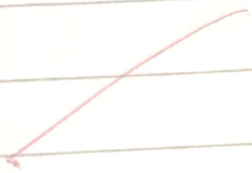
Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself & include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DEF456, you would use:



[$git cherry-pick ABC1231..DEF456]

make sure you are on the branch where you want to apply these changes before running the cherry-pick command.



09 * Analysing and changing Git History *

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date and commit message?

Solution:

To view the details of a specific commit, including the author, date and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1) Using git show

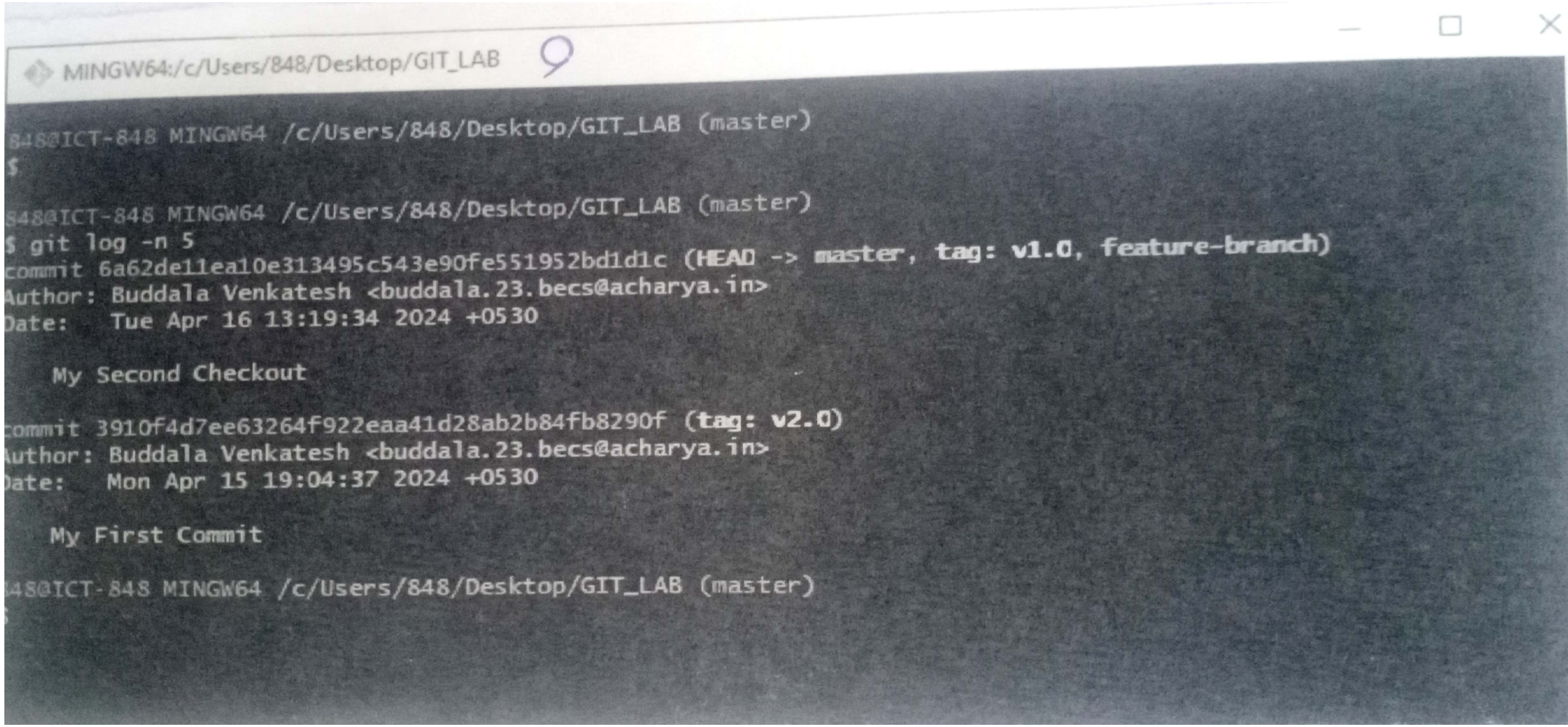
bash

```
git show <commit-ID>
```

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date and the changes introduced by that commit.

For example:-

```
$git show abc123
```



b) Using git log:

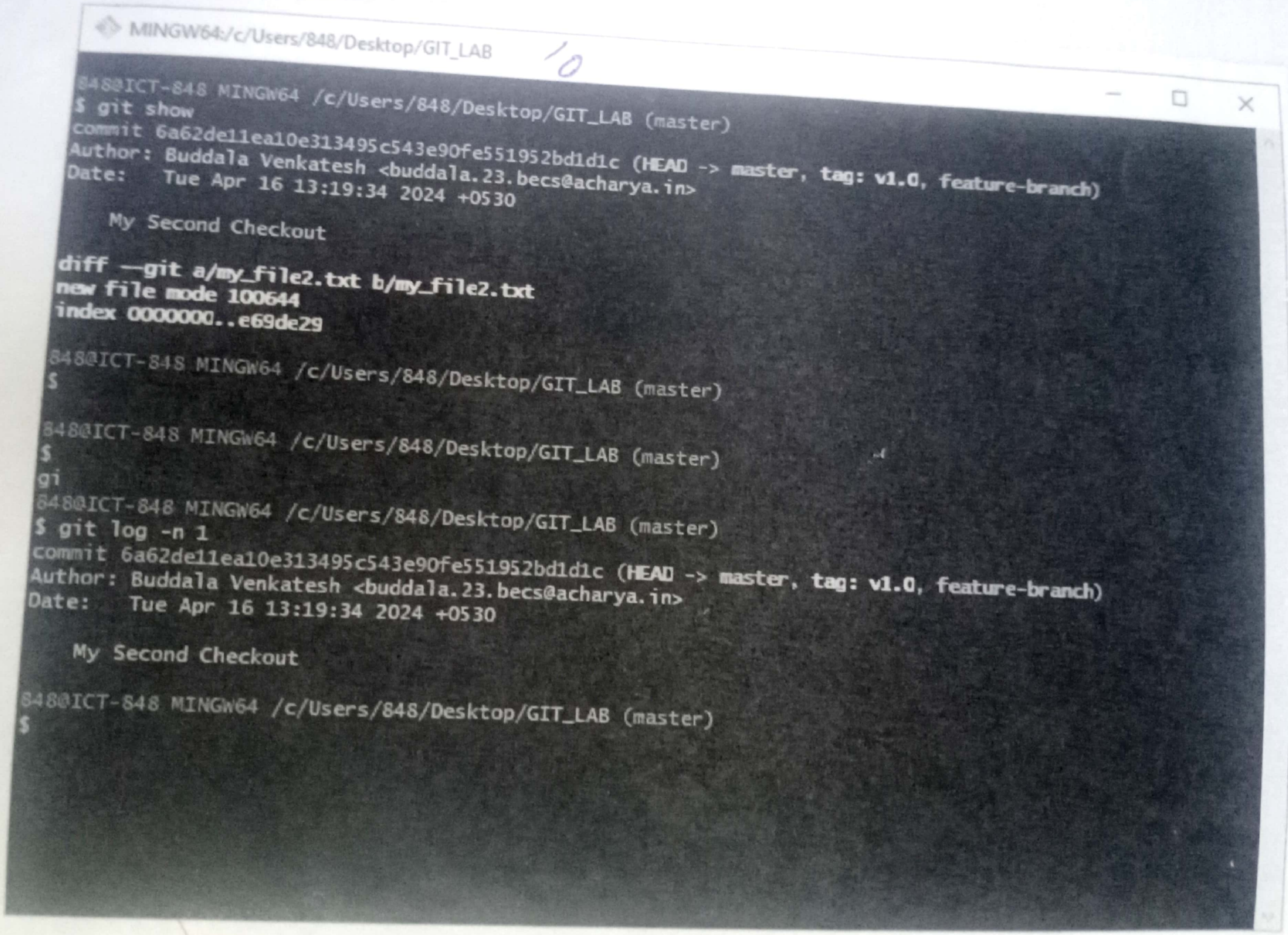
$ git log -n 1 <commit-ID>

The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date and commit ID.

For example:

$ git log -n 1 abc123

Both of these commands will provide you with the necessary information about the specific commit you're interested in.



10 * Analysing and changing Git History *

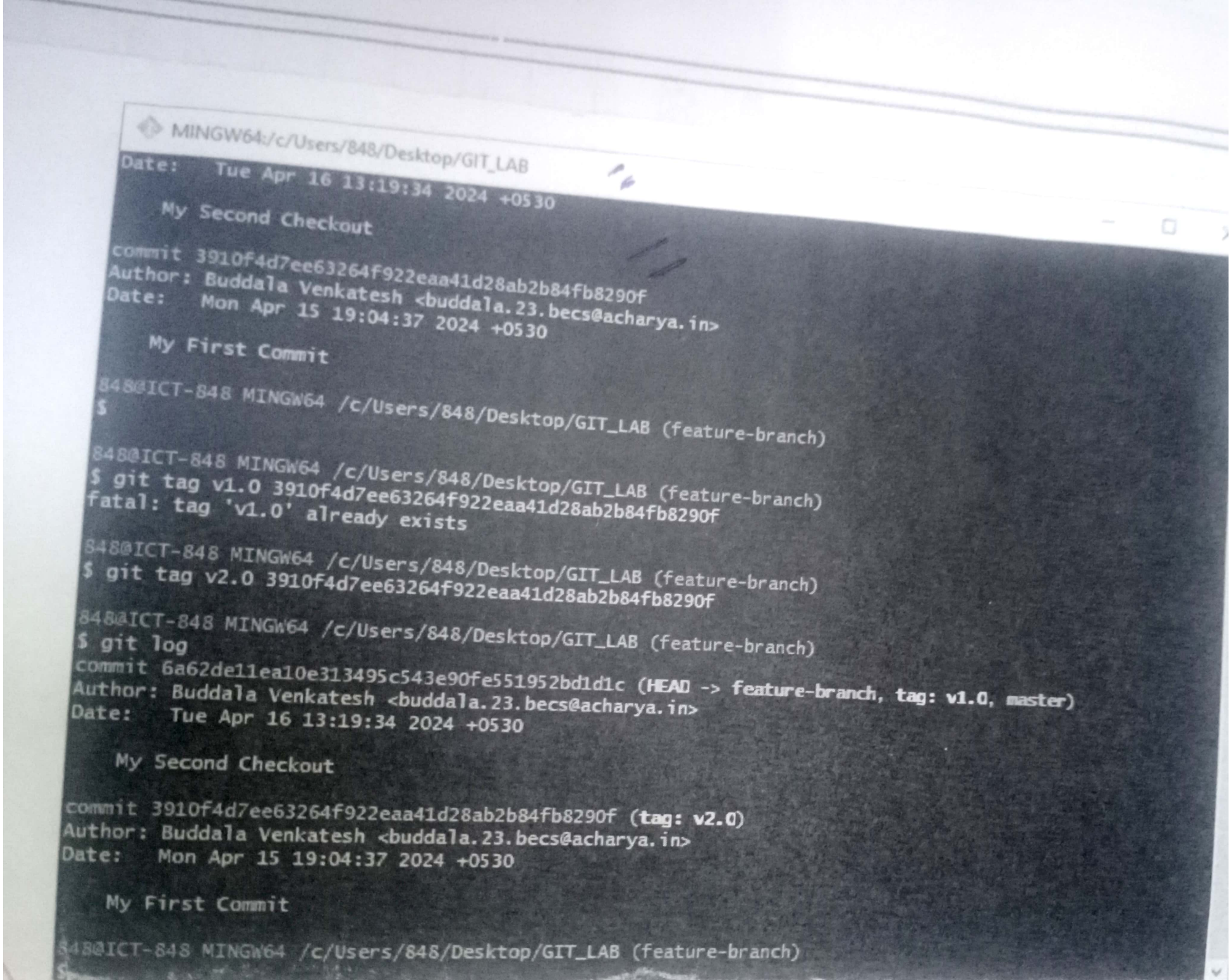
write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31".

Solution:

To list all commit made by the author "JohnDOe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here's the command:

```
$ git
$ git log --author="JohnDoe" --since="2023-01-01" --
  until="2023-12-31"
```

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023 to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.



11 * Analysing and changing Git History *

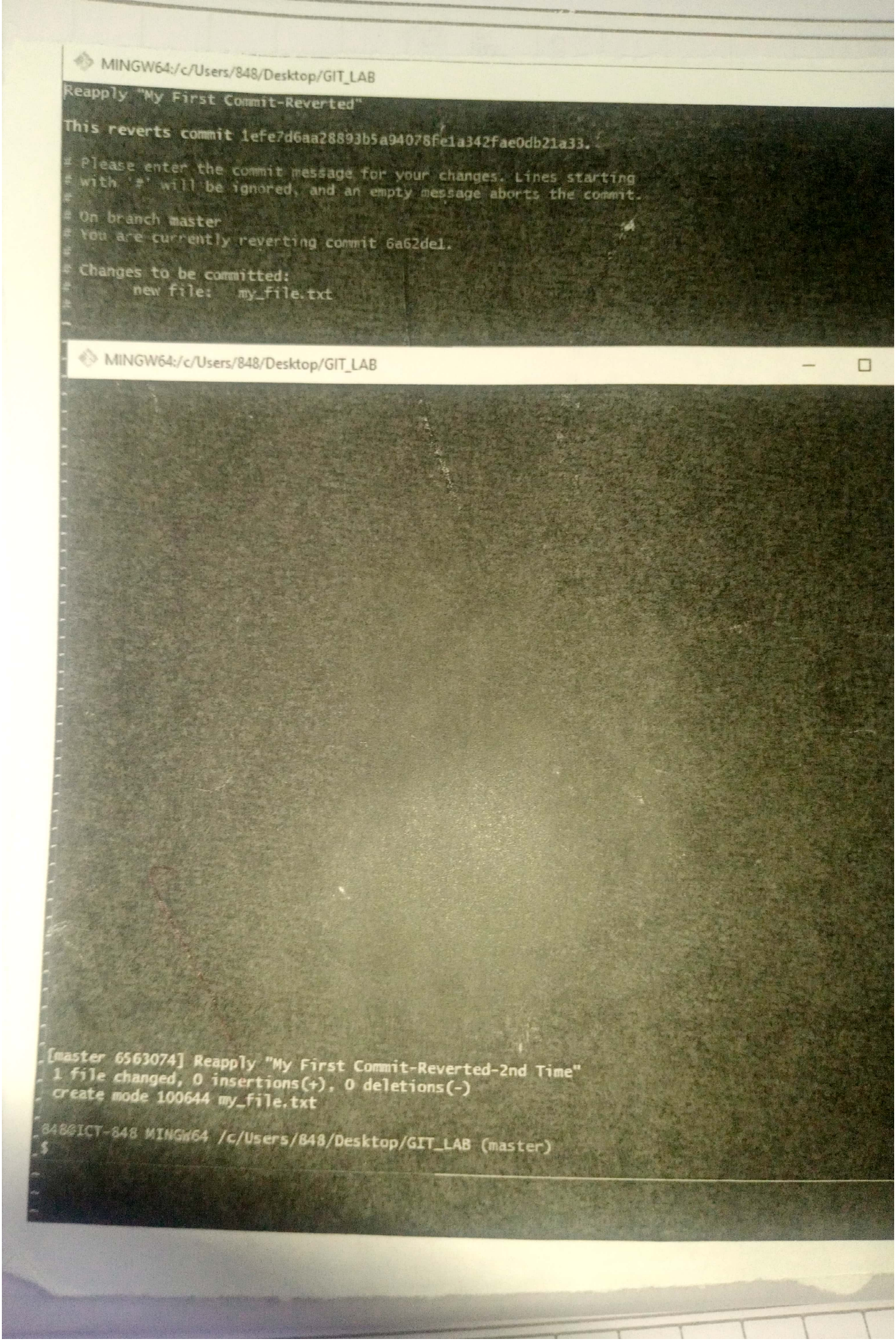
Write the command to display the last five commits in the repository's history.

Solution:

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

[$ git log -n 5]

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.



Experiment No. 12 Date Page No. 20

12 * Analysing and changing Git History *

Write the command to undo the changes introduced by the commit with the ID "abc123"

Solution:

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit. Here's the command.

[$ git revert abc123]

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new commit to record the reversal of the changes.